SQUALL: a Controlled Natural Language for Querying and Updating RDF Graphs

Sébastien Ferré

To cite this version:

Sébastien Ferré. SQUALL: a Controlled Natural Language for Querying and Updating RDF Graphs. Controlled Natural Languages, Aug 2012, Zurich, Switzerland. pp.11-25. hal-00779946

HAL Id: hal-00779946

https://hal.inria.fr/hal-00779946

Submitted on 22 Jan 2013

SQUALL: a Controlled Natural Language for Querying and Updating RDF Graphs

Sébastien Ferré

IRISA, Université de Rennes 1
Campus de Beaulieu, 35042 Rennes cedex, France
ferre@irisa.fr

Abstract. Formal languages play a central role in the Semantic Web. An important aspect regarding their design is syntax as it plays a crucial role in the wide acceptance of the Semantic Web approach. The main advantage of controlled natural languages (CNL) is to reconcile the high-level and natural syntax of natural languages, and the precision and lack of ambiguity of formal languages. In the context of the Semantic Web and Linked Open Data, CNL could not only allow more people to contribute by abstracting from the low-level details, but also make experienced people more productive, and make the produced documents easier to share and maintain. We introduce SQUALL, a controlled natural language for querying and updating RDF graphs. It has a strong adequacy with RDF, an expressiveness close to SPARQL 1.1, and a CNL syntax that completely abstracts from low-level notions such as bindings and relational algebra. We formally define the syntax and semantics of SQUALL as a Montague grammar, and its translation to SPARQL. It features disjunction, negation, quantifiers, built-in predicates, aggregations with grouping, and n-ary relations through reification.

1 Introduction

The Semantic Web [1,7] is founded on a number of formal languages, used to represent: data (RDF), ontologies (RDFS, OWL), rules (SWRL), queries and updates (SPARQL). In this paper, we focus on queries and updates. SPARQL is a very expressive query language [11] as it includes relational algebra, and recently included aggregates, subqueries, negation, and property paths (SPARQL 1.1[1]). It has also been extended with an update language for RDF graphs (SPARQL 1.1 Update[2]). However, its usability is limited because it exhibits low-level notions from relational algebra (e.g., bindings, join, union), and logic (e.g., variables, connectors, quantifiers). An ideal candidate for combining expressiveness and usability is natural language. However, full natural languages have a weak adequacy to Semantic Web formalisms [6], because they are too expressive and too ambiguous. A number of NLP-based systems have been developped for querying the Semantic Web, e.g., Aqualog [9], FREyA [3]. However, those systems

[1] http://www.w3.org/TR/sparql11-query/
[2] http://www.w3.org/TR/sparql11-update/

are generally limited to simple questions (typically equivalent to SPARQL *basic graph patterns*), and cannot be used for updates. For instance, Aqualog queries are limited to two-triples patterns. Therefore, those systems are adequate for simple needs, but are in no way a substitute for formal languages like SPARQL.

Our objective is to define an expressive formal query and update language as a fragment of a natural language. Like Montague, we think that *"There is no important theoretical difference between natural languages and the artificial languages of logicians"* [10]. This opinion is supported by the existence of Controlled Natural Languages (CNL) [13,5]. The main advantage of CNLs is to improve usability by reusing the cognitive capabilities of people, and therefore reducing their learning effort, while retaining the properties of formal languages. To the best of our knowledge, no existing CNL fulfills adequacy and interoperability with RDF and SPARQL. ACE [5] is a general purpose CNL, and requires the definition of a lexicon, which is not available for most RDF datasets (e.g., Linked Open Data). SOS or Rabbit cover OWL ontologies, and also assume some lexical knowledge [12].

In this paper, we introduce SQUALL, a Semantic Query and Update High-Level Language. It qualifies as a CNL for querying and updating RDF graphs. Its contribution is not about expressiveness, which is close to SPARQL. The contribution of SQUALL is to combine (1) an expressiveness close to SPARQL 1.1, (2) a high-level and natural syntax that completely abstracts from low-level notions such as bindings or relational algebra, and (3) a full adequacy with Semantic Web formalisms. SQUALL shares the motivation for a nicer syntax with RDF notations like Turtle and N3, but is rich enough to be a standalone language. The focus of this paper is on the syntax, retaining existing SPARQL notations at the lexical level, i.e., for non-grammatical words (URIs, literals, and variables). This makes SQUALL fully and directly interoperable with existing notations, and this removes all ambiguity at the lexical level. The drawback is that SQUALL sentences may look unnatural because URIs are invariant with respect to number or person: e.g., URI ":Person" stands for "person" and "people", URI ":knows" stands for "knows", "know". Whenever a lexicon, i.e. a non-ambiguous mapping from natural words to URIs and literals, is available or can be produced automatically from an ontology [9], it can easily be integrated into SQUALL to make sentences more natural. In any way, SQUALL has to be learned, like any other formal language. However, we think that it is easier to learn, and that it makes it easier to formulate complex queries and updates: e.g., "for which researcher-s ?X, in graph DBLP every publication whose author is ?X and whose year $\geq$ 2000 has at least 2 author-s ?" is a valid query in SQUALL.

In Section 2, we first give preliminaries about the Semantic Web and Montague grammars. In Section 3, we formally define the syntax and semantics of SQUALL, where semantics is given in terms of an intermediate logical language. Section 4 provides a translation from this intermediate language to SPARQL, therefore providing a concrete semantics as well as a possible implementation of SQUALL as a query and update interface. Section 5 presents our implementation, and illustrates SQUALL and its translation to SPARQL with the above complex query. Section 6 concludes this paper.

2 Preliminaries

We recall basic facts about the Semantic Web and Montague grammars. The Semantic Web provides, through RDF, the data model underlying both SPARQL and SQUALL. Montague grammars provide the theoretical framework in which we formally define the syntax and semantics of SQUALL.

2.1 Semantic Web

The Semantic Web (SW) is founded on several representation languages, such as RDF, RDFS, and OWL, which provide increasing inference capabilities [7]. The two basic units of these languages are *resources* and *triples*. A resource can be either a URI (Uniform Resource Identifier), a literal (e.g., a string, a number, a date), or a *blank node*, i.e., an anonymous resource. A URI is the absolute name of a *resource*, i.e., an entity, and plays the same role as a URL w.r.t. web pages. Like URLs, a URI can be a long and cumbersome string (e.g., `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`), so that it is often denoted by a qualified name (e.g., `rdf:type`), where `rdf:` is the RDF namespace. In the N3 notation, the default namespace `:` can be omitted for qualified names that do not collide with reserved keywords (*bare qualified names*).

A triple ($s\ p\ o$) is made of 3 resources, and can be read as a simple sentence, where s is the subject, p is the verb (called the predicate), and o is the object. For instance, the triple (`Bob friend Alice`) says that "Bob has a friend Alice", where `Bob` and `Alice` are the bare qualified names of two individuals, and `friend` is the bare qualified name of a property, i.e., a binary relation. The triple (`Bob rdf:type man`) says that "Bob has type man", or simply "Bob is a man". Here, the resource `man` is used as a class, and `rdf:type` is a property from the RDF namespace. The triple (`man rdfs:subClassOf person`) says that "man is a subclass of person", or simply "every man is a person". The set of all triples of a knowledge base forms an RDF graph.

Query languages provide on semantic web knowledge bases the same service as SQL on relational databases. They generally assume that implicit triples have been inferred and added to the base. The most well-known query language, SPARQL, reuses the `SELECT FROM WHERE` shape of SQL queries, using graph patterns in the `WHERE` clause. A graph pattern G is one of:

– a triple pattern ($s\ p\ o$.) made of RDF terms and variables (e.g., `?x`),
– a conjunction of two patterns ($\{G_1\ \ G_2\}$),
– an union of two patterns (G_1 `UNION` G_2),
– an optional pattern (`OPTIONAL` G_1),
– a filter pattern (`FILTER` C), where C is a constraint, i.e., either a Boolean expression based on primitive predicates (e.g., comparison, string matching), or a negated graph pattern (`NOT EXISTS` G_1),

- a named graph pattern (`GRAPH` g G_1), where g is URI or a variable denoting a named graph,
- a subquery.

Aggregation operators can be used in the `SELECT` clause (e.g., `COUNT`, `SUM`), and `GROUP BY` clauses can be added to a query. SPARQL has been extended into an update language to insert and delete triples in/from a graph. The most general update form is `INSERT` I `DELETE` D `WHERE` G, where I and D must be sets of triple patterns, and G is a graph pattern that defines bindings for variables occuring in I and D.

2.2 Montague Grammars

Montague grammars [4] are an approach to natural language semantics that is based on formal logic and λ-calculus. It is named after the American logician Richard Montague, who pioneered this approach [10]. A Montague grammar is a context-free generative grammar, where each rule is decorated by a λ-term that denotes the semantics of the syntactic construct defined by the rule. For example, the following rule gives the syntax, semantics, and an example of globally existentially quantified sentences:

$$S \rightarrow \textbf{there is } NP \quad \{ \; np \; \lambda x.\textbf{true} \; \} \quad \text{“there is } [_{NP}\text{a man}]\text{”}$$

Here, **there** and **is** are keywords, or grammatical words, of the language; S (for sentence) and NP (for noun phrase) are syntagms. The semantics is given between curly brackets, and is defined in a fully compositional style, i.e., the semantics of a construct is always a composition of the semantics of sub-constructs. The semantics of a sub-construct is given by the lowercase name of the corresponding syntagm. Every λ-term is typed, and the type of the semantics of a given syntagm is always the same. By convention, we name λ-variables according to their type: x, y, z for RDF *resources* (a.k.a. Montague *entities*), p for RDF properties, t for reified statements (both are special kinds of resources), d for *descriptions*, and s for *sentences*. Sentences are the intention of truth values (a.k.a. Montague *propositions*), and descriptions are the intention of sets of resources (a.k.a. Montague *properties*), i.e., of functions from resources to sentences. In the above rule, x denotes a resource, **true** denotes a sentence, and np denotes the intention of a set of descriptions, i.e., a function from descriptions to sentences (a.k.a. Montague *properties of properties*). Therefore, the whole λ-term denotes a sentence. The constants used in those λ-terms are the constructors of the intermediate language (e.g., **and**, **exists**), and RDF terms. The λ-terms obtained by composition can be simplified according to λ-calculus, through β-reduction (e.g., $(\lambda x.s)\ y =_\beta s[x \leftarrow y]$), and η-expansion (e.g., $d =_\eta \lambda x.(d\ x)$).

3 Syntax and Semantics of SQUALL

In this section, we formally define the syntax and semantics of SQUALL in the style of Montague grammars. This provides a translation from the concrete and natural syntax of SQUALL to an intermediate logical language, rather than directly in terms of an existing query language for the Semantic Web. This is a common practice in the compilation of high-level programming languages, and has a number of advantages. First, it makes the semantics easier to write and understand because defined at a more abstract level. Second, it gives freedom in the choice of the implementation. For instance, the operational semantics of the intermediate language can be given by translating it to an existing language, e.g., SPARQL; by interpreting it in a relational algebra engine; or by using continuation passing-style, like in Prolog. In Section 4, we sketch a solution in the first approach.

We first define a core language corresponding to RDF triples as simple sentences. We then detail a number of orthogonal extensions: relational algebra, queries, quantifiers, reification and n-ary relations, built-in predicates, and aggregations with grouping. The Montague grammars given in this paper provide a self-contained specification of the syntax and semantics of SQUALL.

3.1 Lexical Conventions

The lexical conventions for RDF terms and variables are the same as in Turtle, N3 and SPARQL. In SQUALL examples, we assume the usual namespaces `rdf:`, `rdfs:`, and the default namespace for the domain vocabulary so that bare qualified names can be used for most classes and properties. For reasons of space in SQUALL examples, we will use capital letters (like A, B) instead of full URIs to denote individuals (e.g., publications, persons). The keywords of SQUALL are grammatical words of English: e.g., **is**, **of**, **and**, **every**.

In the scope of this paper, SQUALL directly uses URIs for non-grammatical words, and therefore makes no distinction between singular and plural, nor between nouns and verbs at the lexical level (it does however at the syntactic level). A more essential distinction, which is readily available in RDF schemas, is between unary predicates called *classes*, and binary predicates called *properties*. Therefore, RDF classes are used as nouns (e.g., *woman*) and intransitive verbs (e.g., *work*), while RDF properties are used as relation nouns (e.g., *author*) and transitive verbs (e.g, *know*). For a slightly improved presentation of SQUALL sentences, we allow grammatical suffixes to be appended to URIs: e.g., "work-s" for the third person singular, "author-s" for the plural.

3.2 Triples

A triple (s p o) can be seen as a sentence. The tradition in linguistics [2] is to analyse s and o as noun phrases (NP), p o as a verb phrase (VP), and the whole triple as a sentence (S). In a Semantic Web context, a NP can be a term ($Term$), i.e., one of a URI, a literal, or a variable. A unary predicate ($P1$) can be used as an intransitive verb, and a binary predicate ($P2$) can be used as a transitive verb followed by an object NP. In a Semantic Web context, a $P1$ is typically a class URI, and a $P2$ is typically a property URI. Variables can also be used as predicates.

$$
\begin{array}{lll r}
S & \rightarrow & NP\ VP\ \ \{\ np\ vp\ \} & \text{“}[_{NP}\textsf{A}]\ [_{VP}\textsf{know-s B}]\text{”} \\
NP & \rightarrow & Term\ \ \{\ \lambda d.(d\ term)\ \} & \text{“}\textsf{A}\text{”} \\
VP & \rightarrow & P1\ \ \{\ \lambda x.(p1\ x)\ \} & \text{“}[_{P1}\textsf{work-s}]\text{”} \\
 & | & P2\ NP\ \ \{\ \lambda x.(np\ \lambda y.(p2\ x\ y))\ \} & \text{“}[_{P2}\textsf{know-s}]\ [_{NP}\textsf{B}]\text{”} \\
P1 & \rightarrow & ClassURI\ \ \{\ \lambda x.(\mathbf{type}\ x\ uri)\ \} & \text{“}\textsf{work}\text{”} \\
 & | & Var\ \ \{\ \lambda x.(\mathbf{type}\ x\ var)\ \} & \text{“}\textsf{?C}\text{”} \\
P2 & \rightarrow & PropertyURI\ \ \{\ \lambda x.\lambda y.(\mathbf{stat}\ x\ uri\ y)\ \} & \text{“}\textsf{know}\text{”} \\
 & | & Var\ \ \{\ \lambda x.\lambda y.(\mathbf{stat}\ x\ var\ y)\ \} & \text{“}\textsf{?P}\text{”}
\end{array}
$$

In the semantics, $(\mathbf{stat}\ x\ p\ y)$ denotes a triple statement $x\ p\ y$, and $(\mathbf{type}\ x\ c)$ is a shorthand for $(\mathbf{stat}\ x\ \texttt{rdf:type}\ c)$. We also define $(\mathbf{thing}\ x)$ as a shorthand for $(\mathbf{type}\ x\ \texttt{rdfs:Resource})$. The semantics of a term, when used as a *NP*, is the set of descriptions (λd) of which the term is an instance $(d\ term)$. The semantics of a verb phrase $(P2\ \ NP)$ is the description of resources x such that the description of the resources y that are connected to x through the property $p2$ $(p2\ \ x\ \ y)$, is an instance of np. For instance, the sentence “A know-s B” is parsed as “$[_S[_{NP}A]\ [_{VP}[_{P2}\textsf{know-s}]\ [_{NP}B]]]$”, and translates to $(\lambda d.(d\ A)\ \lambda x.(\lambda d.(d\ B)\ \lambda y.(\mathbf{stat}\ x\ \texttt{know}\ y)))$, which reduces to $(\mathbf{stat}\ A\ \texttt{know}\ B)$. This complexity makes sense when introducing determiners and quantifiers (see Section 3.6).

3.3 Relational Algebra

For queries, SQUALL provides the algebraic operators of SPARQL [11]: **and** for joins, **or** for unions, **not** for differences and negations, **maybe** for optional patterns. Like in Turtle, the coordination **and** can be replaced by a *dot* (**.**) for sentences (S), and by a *comma* (**,**) for *NP*s.

$$
\begin{array}{lll r}
\mathit{\Delta} & \rightarrow & \mathbf{not}\ \mathit{\Delta}_1\ \ \{\ \mathbf{not}\ \delta_1\ \} & \text{“}\textsf{not }[_{VP}\textsf{know-s B}]\text{”} \\
 & | & \mathit{\Delta}_1\ \mathbf{and}\ \mathit{\Delta}_2\ \ \{\ \mathbf{and}\ \delta_1\ \delta_2\ \} & \text{“}[_{VP}\textsf{work-s}]\ \textsf{and}\ [_{VP}\textsf{cite-s X}]\text{”} \\
 & | & \mathit{\Delta}_1\ \mathbf{or}\ \mathit{\Delta}_2\ \ \{\ \mathbf{or}\ \delta_1\ \delta_2\ \} & \text{“}[_{NP}\textsf{A}]\ \textsf{or}\ [_{NP}\textsf{B}]\text{”} \\
 & | & \mathbf{maybe}\ \mathit{\Delta}_1\ \ \{\ \mathbf{option}\ \delta_1\ \} & \text{“}\textsf{maybe }[_{VP}\textsf{know-s B}]\text{”}
\end{array}
$$

In this rule, $\mathit{\Delta}$ stands for any syntagm so that algebraic operators can coordinate all kinds of constructs: e.g., “A or B work-s and cite-s X”, which is equivalent to “A work-s and cite-s X or B work-s and cite-s X”. For a same constructor to apply in constructs having different types, it is necessary to add rewriting rules of λ-terms like the following: $\mathbf{and}\ \delta_1\ \delta_2\ \alpha \rightsquigarrow \mathbf{and}\ (\delta_1\ \alpha)\ (\delta_2\ \alpha)$, where α stands for any argument: e.g., α is an entity x if $\mathit{\Delta} = VP$, α is a description d if $\mathit{\Delta} = NP$.

3.4 Headed NPs, Relatives, and Auxiliary Verbs

This section augments the syntax to make it more natural and flexible. A *NP* can be made of a head preceded by a determiner (*Det*), and followed by an optional term as apposition (*App*), and a coordination of relatives (*Rel*): e.g., “a woman ?A that is an author of X”. The syntagms *NG1* and *NG2* (nominal groups) describe the possible heads, and the relative position of apposition and relatives (*AR*), which are both optional. *Rel* defines the possible relatives. Headed noun phrases allow for new verbal forms based on the auxiliary verbs **is** and **has**: e.g., “is a woman”, “has an author”. The syntagms *NG1*, *AR*, *Rel*, *AP* denote descriptions, the syntagm *NG2* denote binary predicates, while determiners denote binary relations between descriptions (i.e., quantifiers).

$NP \rightarrow Det\ NG1\ \{\ \lambda d.(det\ (\mathbf{init}\ ng1)\ d)\ \}$ “[$_{Det}$a] [$_{NG1}$woman]”
$\mid\ Det\ NG2\ \mathbf{of}\ NP\ \{\ \lambda d.(np\ \lambda x.(det\ (\mathbf{init}\ (ng2\ x))\ d))\ \}$
“[$_{Det}$the] [$_{NG2}$author-s] of [$_{NP}$X]”
$Det \rightarrow \mathbf{a(n)}\ \{\ \lambda d_1.\lambda d_2.(\mathbf{exists}\ (\mathbf{and}\ d_1\ d_2))\ \}$
$\mid\ \mathbf{the}\ \{\ \lambda d_1.\lambda d_2.(\mathbf{the}\ d_1\ d_2)\ \}$
$NG1 \rightarrow \mathbf{thing}\ AR\ \{\ \mathbf{and\ thing}\ ar\ \}$ “thing [$_{AR}$that cite-s A]”
$\mid\ P1\ AR\ \{\ \mathbf{and}\ p1\ ar\ \}$ “[$_{P1}$woman] [$_{AR}$?A]”
$NG2 \rightarrow P2\ AR\ \{\ \lambda x.\lambda y.(\mathbf{and}\ (p2\ x\ y)\ (ar\ y))\ \}$
“[$_{P2}$author] [$_{AR}$?A]”
$AR \rightarrow App\ Rel\ \{\ \mathbf{and}\ app\ rel\ \}$ “[$_{App}$?A] [$_{Rel}$that X cite-s]”
$\mid\ App\ \{\ app\ \}$
$App \rightarrow URI\ \{\ \lambda x.(\mathbf{eq}\ x\ uri)\ \}$ “A”
$\mid\ Var\ \{\ \lambda x.(\mathbf{bind}\ x\ var)\ \}$ “?X”
$\mid\ \epsilon\ \{\ \lambda x.\mathbf{true}\ \}$
$Rel \rightarrow \mathbf{that}\ VP\ \{\ \mathbf{init}\ vp\ \}$ “that [$_{VP}$know-s B]”
$\mid\ \mathbf{that}\ NP\ P2\ \{\ \mathbf{init}\ \lambda x.(np\ \lambda y.(p2\ y\ x))\ \}$
“that [$_{NP}$X] [$_{P2}$cite-s]”
$\mid\ \mathbf{such\ that}\ S\ \{\ \mathbf{init}\ \lambda x.s\ \}$ “such that [$_{S}$?A work-s]”
$\mid\ Det\ NG2\ \mathbf{of\ which}\ VP\ \{\ \mathbf{init}\ \lambda x.(det\ (ng2\ x)\ vp)\ \}$
“[$_{Det}$an] [$_{NG2}$author] of which [$_{VP}$know-s B]”
$\mid\ \mathbf{whose}\ NG2\ VP \equiv \mathbf{the}\ NG2\ \mathbf{of\ which}\ VP$
“whose [$_{NG2}$author ?A] [$_{VP}$cites-s a colleague of ?A]”
$\mid\ \mathbf{whose}\ P2\ \mathbf{is/are}\ NP\ \{\ \lambda x.(np\ \lambda y.(p2\ x\ y))\ \}$
“whose [$_{P2}$author] [$_{VP}$is a woman]”
$VP \rightarrow \mathbf{is/are}\ AP\ \{\ ap\ \}$ “is [$_{AP}$a woman]”
$\mid\ \mathbf{is/are}\ Rel\ \{\ rel\ \}$ “is [$_{Rel}$such that ?A work-s]”
$\mid\ \mathbf{has/have}\ Det\ P2\ AR\ \{\ \lambda x.(det\ (p2\ x)\ ar)\ \}$
“have [$_{Det}$an] [$_{P2}$author] [$_{AR}$that X cite-s]”
$AP \rightarrow Term\ \{\ \lambda x.(\mathbf{eq}\ x\ term)\ \}$ “A”
$\mid\ \mathbf{a(n)/the}\ NG1\ \{\ ng1\ \}$ “a [$_{NG1}$woman]”
$\mid\ \mathbf{a(n)/the}\ NG2\ \mathbf{of}\ NP\ \{\ \lambda x.(np\ \lambda y.(ng2\ y\ x))\ \}$
“the [$_{NG2}$author] of [$_{NP}$X]”
$S \rightarrow S_1\ \mathbf{where}\ S_2\ \{\ \mathbf{where}\ s_1\ s_2\ \}$
“[$_{S}$A know-s ?B] where [$_{S}$?B is a woman]”

The meaning of the function **init** is clarified in Section 3.7 about reification. At this point, it can be assumed to be the identity function. The constructor **exists** is an existential quantification over an entity. The constructor **eq** represents equality (a binary predicate), and the constructor (**bind** x y) represents an assignement of x into y. The keywords **the** and **where** are propagated to the semantics using the constructors **the** and **where**, because their interpretation differs whether they occur in the scope of a query or an update (see Section 4). In short, they are respectively equivalent to **a** and **and** in queries, and separate conditions and changes in updates.

3.5 Queries

Question words distinguish declarative sentences (updates) from interrogative sentences (queries): **whether** introduces a closed question, whose semantics encapsulates the sentence in the constructor **ask**; **what** can be used in place of any *NP* to form open questions. The constructor **select** applies to a description, and indicates that the instances of this description should be returned as query results. Multi-dimensional queries are expressed by using several occurences of **what**: e.g., "what is the author of what". Some question words are used as determiners. **which** has the same effect as **what** while allowing a restriction on returned resources: e.g., "which woman is an author of X". **how many** provides the easy expression of the most common aggregation, counting: e.g., "how many person is an author of X". The constructor **count** applies to a description, and indicates that the number of its instances should be returned as a query result.

$$\begin{array}{lll}
S & \rightarrow \textbf{whether } S_1 \;\; \{\, \textbf{ask } s_1 \,\} & \text{“whether } [_S\text{A know-s a woman}]\text{”} \\
NP & \rightarrow \textbf{what} \equiv \textbf{which thing} & \\
 & \mid\; \textbf{whose } \mathit{NG2} \equiv \textbf{the } \mathit{NG2} \textbf{ of what} & \\
 & & \text{“whose } [_{NG2}\text{author ?A}]\text{”} \\
Det & \rightarrow \textbf{which} \;\; \{\, \lambda d_1.\lambda d_2.(\textbf{select } (\textbf{and } d_1\ d_2)) \,\} & \\
 & \mid\; \textbf{how many} \;\; \{\, \lambda d_1.\lambda d_2.(\textbf{count } (\textbf{and } d_1\ d_2)) \,\} &
\end{array}$$

In order for the question constructors not to appear in the scope of non-question constructors, we introduce rewriting rules like the following: **and** (**select** $\lambda x.s_1$) $s_2 \rightsquigarrow$ **select** $\lambda x.$(**and** s_1 s_2); **exists** $\lambda x.$(**select** $\lambda y.s$) $\rightsquigarrow$ **select** $\lambda y.$(**exists** $\lambda x.s$). Similar rules are to be defined for **count**, other algebraic operators (like **and**) and other quantifiers (like **exists**), which are presented in Section 3.3 and 3.6. Also, we ensure that no **select** falls in the scope of **count** by adding the additional rewriting rule: **count** $\lambda x.$(**select** $\lambda y.s$) $\rightsquigarrow$ **select** $\lambda y.$(**count** $\lambda x.s$).

A sentence is a valid query if its semantics contains either one **ask** or any number of **select** and possibly one **count**. A sentence is a valid update if its semantics contains none of the question constructors. Otherwise, the sentence is invalid, and a syntax error should be returned.

3.6 Quantifiers

Quantifiers are commonplace in natural languages in the form of determiners, whereas they are notoriously difficult to express in SPARQL or SQL [8]. The quantifier **exists** applies to a description, and checks that its extension is not empty. The quantifier **forall** applies to two descriptions, and checks that the extension of the first is included in the extension of the second. The constructor **atleast** i applies to a description, and checks that its extension has at least i elements. A possible use of them is: "every person is an author of at least 10 publication-s".

$$\begin{array}{lll} Det & \rightarrow & \textbf{some} \;\; \{ \; \lambda d_1.\lambda d_2.(\textbf{exists}\ (\textbf{and}\ d_1\ d_2)) \; \} \\ & | & \textbf{every} \;\; \{ \; \lambda d_1.\lambda d_2.(\textbf{forall}\ d_1\ d_2) \; \} \\ & | & \textbf{no} \;\; \{ \; \lambda d_1.\lambda d_2.(\textbf{not}\ (\textbf{exists}\ (\textbf{and}\ d_1\ d_2))) \; \} \\ & | & \textbf{at least}\ i \;\; \{ \; \lambda d_1.\lambda d_2.(\textbf{atleast}\ i\ (\textbf{and}\ d_1\ d_2)) \; \} \\ S & \rightarrow & \textbf{for}\ NP\ ,\ S \;\; \{ \; np\ \lambda x.s \; \} \\ & | & \textbf{there is}\ NP \;\; \{ \; np\ \lambda x.\textbf{true} \; \} \end{array}$$

The grammar rules are defined so that the scope of quantifiers are leftmost-outermost, and are restricted to the scope of the related verb. Therefore, "every man love-s some woman" means there is possibly a different woman for each man; while in "there is a woman that every man love-s" means there is a single woman.

The keywords **for** and **there is** introduce global quantifiers, in a style closer to mathematical logic. Their scope extends to the end of the sentence, which may be a coordinated sentence: e.g., "for every publication ?X, ?X has an author ?A and ?A cite-s ?X". Relatives introduced by **such that** can be used in a global existential quantification: "there is a person ?X such that no publication has the author ?X".

3.7 Reification and N-ary Predicates

RDF triples can be reified as statements. Therefore, it is useful to allow statements about statements. The keyword **that** turns a sentence into a noun phrase, changing the focus from the truth of the sentence to the statements involved in the sentence. The variable t is used for denoting individual statements. Here it becomes necessary to explicit two arguments shared by all constructs, and that were left implicit so far (thanks to η-equivalence of λ-calculus): $\overline{a}$ is a list of additional arguments to the predicate of the sentence, and g denotes a graph, i.e., a set of statements. Together, they form the context in which a sentence is interpreted. Contexts are the counterpart of worlds in Montague semantics. The extension of a phrase is obtained by passing a context, and an intension is formed by abstracting over a context.

In the sentence "A say-s that B is an author of X", the object of the property "say" is the statement of the triple (X `author` B). In the semantics, the main clause (verb "say-s") abstracted over its object and receiving its own context ($\lambda t.$(**stat** A `say` $t\ \overline{a}\ g$)) denotes a graph (i.e., a set of statements), and is passed as a context to the dependent clause (**stat** X `author` B). This generalizes the GRAPH construct in SPARQL, where a graph would be restricted to the form $\lambda t.$(**stat** t `graph` G), where the triple (t `graph` G) would mean that the statement t is a member of ("has graph") the named graph G (a URI or a variable).

The other rules define prepositions ($Prep$) and prepositional phrases (PP), and their inclusion in sentences. A PP is introduced by **at** or **in**, and is followed by a *Prep* and a *NP*. It modifies a sentence by adding an argument to its context (constructor **arg**), or by encapsulating it in a graph (constructor **graph**). Each argument is a pair $(uri, term)$, where *uri* defines the role of the argument w.r.t. the statement, much like “rdf:subject”, “rdf:predicate”, and “rdf:object”; and *term* is the argument itself. Those arguments provide means for expressing n-ary relationships. For example, “A is an author of X at rank 1” means that A is the first author of X; and “A sell-s X in year 2009 at amount 10” means that A sold 10 units of X in 2009.

The remaining rules say that prepositional phrases can occur anywhere in a sentence, and the syntagms *OP* (object phrase) and *CP* (complement phrase) are introduced to retain the leftmost-outermost scope rule for quantifiers. Therefore, the rigidity of this scope rule is balanced by the free ordering of *PP*s, the possible inversion between subject and object (**of**), and the global quantifiers introduced by **for**. An example of query that uses a number of those constructs is: “at which venue every professor that work-s at place X speak-s at some time”.

$NP \rightarrow$ **that** S $\{\ \lambda d.\lambda \overline{a}.\lambda g.(s\ ()\ \lambda t.(d\ t\ \overline{a}\ g))\ \}$

“that [$_S$A know-s B]”

$PP \rightarrow$ **at/in** $Prep\ NP$ $\{\ \lambda s.(np\ \lambda z.(prep\ z\ s))\ \}$

“at place [$_{NP}$the city Rennes]”

$\mid$ **at/in** $Det\ Prep\ AR \equiv$ **at** $Prep\ Det$ **thing** AR

“at [$_{Det}$some] venue [$_{AR}$whose place is Rennes]”

$S \rightarrow PP\ S$ $\{\ pp\ s\ \}$

$VP \rightarrow PP\ VP$ $\{\ \lambda x.(pp\ (vp\ x))\ \}$

$\mid P1\ CP$ $\{\ \lambda x.(cp\ (p1\ x))\ \}$

$\mid P2\ OP$ $\{\ \lambda x.(op\ (p2\ x))\ \}$

$\mid$ **is/are** $AP\ CP$ $\{\ \lambda x.(cp\ (ap\ x))\ \}$

$\mid$ **has/have** $Det\ P2\ AR\ CP$ $\{\ \lambda x.(det\ \lambda y.(cp\ (p2\ x\ y))\ ar)\ \}$

$OP \rightarrow PP\ OP$ $\{\ \lambda d.(pp\ (op\ d))\ \}$

$\mid NP\ CP$ $\{\ \lambda d.(np\ \lambda y.(cp\ (d\ y)))\ \}$

$CP \rightarrow PP\ CP$ $\{\ \lambda s.(pp\ (cp\ s))\ \} \mid \epsilon$ $\{\ \lambda s.s\ \}$

$Rel \rightarrow$ **at/in which** $Prep\ AR\ S$ $\{$ **init** $\lambda x.($**and** $(ar\ x)\ (prep\ x\ s))\ \}$

“in which graph [$_S$A work-s]”

$Prep \rightarrow$ **graph** $\{$ **graph** $\} \mid URI$ $\{$ **arg** $uri\ \}$

It is now possible to clarify and define the constructors **init** and **arg** that have already been used. The constructor **init** reinitializes the list of arguments in some construct: **init** $= \lambda d.\lambda x.\lambda \overline{a}.(d\ x\ ())$. This is useful to restrict the passing of arguments to the predicate of a sentence, and to avoid its propagation to noun groups and relatives for instance. The constructor **arg** adds an argument to the current list of arguments, waiting to be passed to the main predicate of the sentence: **arg** $= \lambda uri.\lambda z.\lambda s.\lambda \overline{a}.(s\ ((uri, z), \overline{a}))$. **graph** is a special preposition that represents the membership of statements to SPARQL named graphs.

In order to pass the context down to arguments of algebraic operators, quantifiers, and question constructors, it is necessary to define rewriting rules like the following: **true** $\overline{a}$ $g \rightsquigarrow$ **true**, **and** s_1 s_2 $\overline{a}$ $g \rightsquigarrow$ **and** $(s_1\ \overline{a}\ g)\ (s_2\ \overline{a}\ g)$, **exists** $d\ \overline{a}\ g \rightsquigarrow$ **exists** $\lambda x.(d\ x\ \overline{a}\ g)$.

3.8 Built-in Predicates and Aggregations

Built-in predicates are used in SPARQL constraints (e.g., comparison, arithmetic, string matching), and expect a variable number of arguments. They can be used as nouns or intransitive verbs (*Pred1URI*), when one argument plays the role of a subject, and as relation noun or transitive verbs (*Pred2URI*), when two arguments play the role of subject and object. Additional arguments are represented through prepositional phrases, assigning a different preposition URI to each. For example, assuming the binary predicate "match" for regular expression matching, and the unary predicate "Monday" denoting the set of Monday dates, the following query can be expressed: "which woman has a lastname that match-es "Smi.*" and a birth whose date is a Monday".

The head of a noun phrase (*NG1*) can be an aggregator followed by the set of what should be aggregated, and optionally followed by a list of grouping dimensions introduced by **per**. Each dimension is an *AP* that specifies the set of possible values for this dimension: e.g., "what is the count of the publication-s ?P per the year of ?P, and the affiliation of an author of ?P". This construction produces bindings for the aggregated value and each dimension.

$P1 \rightarrow Pred1URI\ \{\ \lambda x.(\mathbf{pred1}\ uri\ x)\ \}$ "Monday"
$P2 \rightarrow Pred2URI\ \{\ \lambda x.\lambda y.(\mathbf{pred2}\ uri\ x\ y)\ \}$ "match"
$NG1 \rightarrow AggregURI\ \mathbf{of}\ AP\ (\mathbf{per}\ AP_i^+)?\ \{\ \lambda x.(\mathbf{aggreg}\ uri\ x\ ap\ (ap_i)_i)\ \}$
"count of [$_{AP}$the publication ?P] per [$_{AP}$the year of ?P]"

3.9 Handling of Ambiguity

The price for the natural and flexible syntax of SQUALL is ambiguity, i.e., the fact that some sentences can be parsed in different ways possibly leading to different semantics. In SQUALL, ambiguities are resolved by the following rules:

1. when forming a construct Δ from one or two constructs of same syntagm Δ (e.g., coordinating 2 *NP*s, modifying a sentence with a *PP*), algebraic operators have priority (in decreasing priority order: **not**, **maybe**, **and**, **or**, **where**) over sentence modifiers (*PP* as a prefix, and global quantifiers **for** *NP*). Punctuation has lowest priority, and right-associativity is used for binary coordinations;
2. smaller syntagms have priority over larger syntagms, i.e., in decreasing priority order: *P1*, *P2*, *Det*, *Rel*, *NG1*, *NG2*, *AP*, *NP*, *PP*, *CP*, *OP*, *VP*, *S*;
3. in case of ambiguity between forming two constructs of same syntagm, the shorter construct is chosen.

Round brackets can be used for every syntagms to escape those rules. Rule 2 implies that "a man or woman" is interpreted as "a (man or woman)" rather than "(a man) or woman", as *NG1* has priority over *NP*. Rule 3 implies that in "*A* know-s a researcher that *X* cite-s at venue *V*", the PP "at venue *V*" binds to the shorter VP "cite-s ..." rather than to the longer VP "know-s ...".

4 Translation to SPARQL

An operational semantics can be given to SQUALL by translating its intermediate language to SPARQL. Given a SQUALL sentence S with semantics s, the translation of S in the intermediate language is the formula $f = s\ ()\ g_0$, i.e., the sentence initialized with an empty list of arguments, and some default graph g_0 (e.g., $\lambda t.\mathbf{true}$). In order to simplify this formula, we remove some of the constructors of the intermediate language by giving them a definition. For example, the triple $(x\ p\ y)$ passed to the constructor **stat** is reified by stating the existence of a statement resource t, which is connected to its subject, predicate, object, and arguments through roles. Each connection is represented using the new constructor **triple** that represents a non-reifiable triple. Those connections are completed by stating that the statement t belongs to the given graph g. This can be simplified into **triple** $x\ p\ y$ when there are no argument, and the graph is $\lambda t.\mathbf{true}$. The constructor **eq** is defined by the built-in predicate $=$. The constructors **atleast** and **count** are defined in terms of the aggregator `COUNT`, and the built-in predicate $\geq$. The constructor **fold** is the classical iterator on lists. A term $(\mathbf{fold}\ f\ e\ (x, \overline{x}))$ reduces to $(\mathbf{fold}\ f\ (f\ e\ x)\ \overline{x})$, and a term $(\mathbf{fold}\ f\ e\ ())$ reduces to e.

$$\begin{aligned}
&\mathbf{stat} = \lambda x.\lambda p.\lambda y.\lambda \overline{a}.\lambda g.(\mathbf{exists}\ \lambda t.(\mathbf{fold}\\
&\quad \lambda q.\lambda(uri, z).(\mathbf{and}\ q\ (\mathbf{triple}\ t\ uri\ z))\ (g\ t)\\
&\quad ((\text{rdf:subject}, x), (\text{rdf:predicate}, p), (\text{rdf:object}, y), \overline{a})))\\
&\mathbf{eq} = \lambda x.\lambda y.(\mathbf{pred2}\ (=)\ x\ y)\\
&\mathbf{atleast} = \lambda i.\lambda d.(\mathbf{exists}\ \lambda x.(\mathbf{and}\ (\mathbf{aggreg}\ \texttt{COUNT}\ x\ d\ ())\ (\mathbf{pred2}\ \geq\ x\ i)))\\
&\mathbf{count} = \lambda d.(\mathbf{select}\ \lambda x.(\mathbf{aggreg}\ \texttt{COUNT}\ x\ d\ ()))
\end{aligned}$$

After applying those definitions, the only remaining constructors are: **triple**, **bind**, **pred1**, **pred2**, **aggreg**, **true**, **not**, **and**, **or**, **option**, **where**, **exists**, **forall**, **the**, **ask**, **select**. We define in the following their translation to the queries and updates of SPARQL. Those translations are chosen to be concise, and not to be optimal in any way. The SPARQL translation of a formula f is denoted by $[f]$, and $[X \mid f]_Q$ is an auxialiary translation for multi-dimensional queries. The two other auxiliary translations are $[f]_G$ for producing graph patterns (that generate bindings), and $[f]_U$ for producing updates (that insert and delete triples). An update is a triple (I, D, G), where I is a graph to be inserted, D is a graph to be deleted, and G is a graph pattern.

$$
\begin{array}{l}
[\textbf{ask}\ f] = \texttt{ASK}\ \{\ [f]_G\ \} \\
[\textbf{select}\ d] = [?x \mid d\ ?x]_Q \\
[f] = \texttt{INSERT}\ \{I\}\ \texttt{DELETE}\ \{D\}\ \texttt{WHERE}\ \{\ G\ \} \quad \text{where } (I, D, G) = [f]_U \\
[X \mid \textbf{select}\ d]_Q = [X\ ?x \mid d\ ?x]_Q \\
[X \mid f]_Q = \texttt{SELECT}\ X\ \texttt{WHERE}\ \{\ [f]_G\ \}
\end{array}
$$

Formulas with constructors **ask** and **select** translate to the corresponding SPARQL queries, while other formulas translate to SPARQL updates. Every occurence of a SPARQL variable $?x$ assumes the generation of a fresh variable name. Those variables are used to instantiate description parameters of question, quantifier, and aggregation constructors.

$$
\begin{array}{l}
[\textbf{triple}\ s\ p\ o]_G = s\ p\ o\ . \\
[\textbf{bind}\ x\ y\ \overline{a}\ g]_G = \texttt{BIND}\ (\ x\ \texttt{AS}\ y\ ) \\
[\textbf{pred1}\ \mathit{pred}\ x\ \overline{a}\ g]_G = \texttt{FILTER}\ \mathit{pred}(x, \overline{a}) \\
[\textbf{pred2}\ \mathit{pred}\ x\ y\ \overline{a}\ g]_G = \texttt{FILTER}\ \mathit{pred}(x, y, \overline{a}) \\
[\textbf{graph}\ x\ s\ \overline{a}\ g]_G = \texttt{GRAPH}\ x\ \{\ [s\ ()\ \lambda t.\textbf{true}]_G\ \} \\
[\textbf{aggreg}\ \mathit{agg}\ x\ d\ (d_i)_i\ \overline{a}\ g]_G = \\
\quad \{\ \texttt{SELECT}\ (?z_i)_i\ (\ \mathit{agg}(?y)\ \text{AS}\ ?x\ ) \\
\quad\ \ \texttt{WHERE}\ \{\ [\textbf{fold}\ (\textbf{and})\ (d\ ?y)\ (d_i\ ?z_i)_i]_G\ \} \\
\quad\ \ \texttt{GROUP BY}\ (?z_i)_i\ \} \\
[\textbf{true}]_G = \epsilon \\
[\textbf{and}\ f_1\ f_2]_G = [f_1]_G\ [f_2]_G \\
[\textbf{or}\ f_1\ f_2]_G = \{\ [f_1]_G\ \}\ \texttt{UNION}\ \{\ [f_2]_G\ \} \\
[\textbf{option}\ f]_G = \texttt{OPTIONAL}\ \{\ [f]_G\ \} \\
[\textbf{not}\ f]_G = \texttt{FILTER NOT EXISTS}\ \{\ [f]_G\ \} \\
[\textbf{where}\ f_1\ f_2]_G = [\textbf{and}\ f_1\ f_2]_G \\
[\textbf{exists}\ d]_G = [d\ ?x]_G \\
[\textbf{forall}\ d_1\ d_2]_G = [\textbf{not}\ (\textbf{exists}\ (\textbf{and}\ d_1\ (\textbf{not}\ d_2)))]_G \\
[\textbf{the}\ d_1\ d_2]_G = [\textbf{exists}\ (\textbf{and}\ d_1\ d_2)]_G
\end{array}
$$

Built-in predicates translate to SPARQL filters, and aggregations translate to SPARQL aggregative sub-queries. Arguments can be used for n-ary predicates, but there is no counterpart in SPARQL for aggregations. Algebraic constructors translate to their SPARQL counterpart, and quantifiers all translate to the implicit SPARQL existential quantifier and negation.

$$
\begin{array}{l}
[\textbf{triple}\ s\ p\ o]_U = (s\ p\ o\ ., \epsilon, \epsilon) \\
[\textbf{true}]_U = (\epsilon, \epsilon, \epsilon) \\
[\textbf{and}\ f_1\ f_2]_U = (I_1\ I_2, D_1\ D_2, G_1\ G_2) \\
[\textbf{not}\ f]_U = (D, I, G) \quad \text{where } (I, D, G) = [f]_U \\
[\textbf{where}\ f_1\ f_2]_U = (I_1, D_1, G_1\ [f_2]_G) \quad \text{where } (I_1, D_1, G_1) = [f_1]_U \\
[\textbf{exists}\ d]_U = [d\ ?x]_U \\
[\textbf{forall}\ d_1\ d_2]_U = [\textbf{where}\ (d_2\ ?x)\ (d_1\ ?x)]_U \\
[\textbf{the}\ d_1\ d_2]_U = [\textbf{where}\ (d_2\ ?x)\ (d_1\ ?x)]_U
\end{array}
$$

Compare the translation of **where** with graph patterns. In updates, it introduces a graph pattern, whereas in a graph pattern, it refines it like **and**. The same can be said for the quantifier **the**.

5 Implementation and Illustration

The contents of this paper has been implemented in less than 500 lines of OCaml[3], a functional language. The code is very close to the formalism used in this paper, which makes it easy to extend the theory and the code in parallel. The source code, and a SPARQL translator Web form are available from the SQUALL web page at `http://www.irisa.fr/LIS/softwares/squall`.

As an illustration of the translation to SPARQL, we consider the sentence given in the introduction: "for which researcher-s ?X, in graph DBLP every publication whose author is ?X and whose year $\geq$ 2000 has at least 2 author-s". Its syntactic analysis is

> "$[_S$for $[_{NP}[_{Det}$which$]$ $[_{NG1}[_{P1}$researcher-s$]$ $[_{AR}[_{App}$?X$]]]]$, $[_S[_{PP}$in $[_{Prep}$graph$]$ $[_{NP}$DBLP$]]$ $[_S[_{NP}[_{Det}$every$]$ $[_{NG1}[_{P1}$publication$]$ $[_{AR}[_{Rel}[_{Rel}$whose $[_{NG2}[_{P2}$author$]]$ $[_{VP}$is $[_{AP}$?X$]]]$ and $[_{Rel}$whose $[_{NG2}[_{P2}$year$]]$ $[_{VP}[_{P2}\geq]$ $[_{NP}$2000$]]]]]]]$ $[_{VP}$has $[_{Det}$at least 2$]$ $[_{P2}$author-s$]]]]]$",

and its SPARQL translation is as follows (with some reformatting, and un-reifying triples whose reification is not used):

```
SELECT ?r
WHERE {
    ?r rdf:type :researcher .
    BIND (?r AS ?X)
    GRAPH :DBLP {
        FILTER NOT EXISTS {
            ?p rdf:type :publication .
            ?p :author ?X .
            ?p :year ?y .
            FILTER (?y >= 2000)
            FILTER NOT EXISTS {
                { SELECT COUNT(?a) AS ?n
                  WHERE { ?p :author ?a . } }
                FILTER (?n >= 2) } } } }
```

6 Conclusion

SQUALL is a Semantic Query and Update High-Level Language that provides a controlled natural language on top of SPARQL 1.1, while preserving adequacy,

[3] `http://caml.inria.fr/ocaml/`

interoperability, and expressiveness. Its syntax and semantics are formally defined as a Montague grammar. The semantics of SQUALL sentences are formulated in a logical intermediate language. We have sketched a translation from this intermediate language to SPARQL, thus providing an operational semantics and a possible implementation for SQUALL. Possible future work include the addition of natural language constructs (e.g., arithmetic and string expressions, comparatives and superlatives, anaphoras other than variables), the full coverage of SPARQL 1.1 (e.g., expressions, property paths), and the use of ontologies and lexicons to improve the grammaticality of the language.

References

1. Antoniou, G., van Harmelen, F.: A Semantic Web Primer. MIT Press (2004)
2. Biber, D., Johansson, S., Leech, G., Conrad, S., Finegan, E.: Longman grammar of spoken and written English. Pearson Education Limited (1999)
3. Damljanovic, D., Agatonovic, M., Cunningham, H.: Identification of the question focus: Combining syntactic analysis and ontology-based lookup through the user interaction. In: Language Resources and Evaluation Conference (LREC). ELRA (2010)
4. Dowty, D.R., Wall, R.E., Peters, S.: Introduction to Montague Semantics. D. Reidel Publishing Company (1981)
5. Fuchs, N.E., Kaljurand, K., Schneider, G.: Attempto Controlled English meets the challenges of knowledge representation, reasoning, interoperability and user interfaces. In: Sutcliffe, G., Goebel, R. (eds.) FLAIRS Conference. pp. 664–669. AAAI Press (2006)
6. Haase, P., Broekstra, J., Eberhart, A., Volz, R.: A comparison of RDF query languages. In: *et al.*, S.M. (ed.) Int. Semantic Web Conf. pp. 502–517. LNCS 3298, Springer (2004)
7. Hitzler, P., KrÃ¶tzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC (2009)
8. Hsu, P.Y., Jr., D.S.P.: Improving SQL with generalized quantifiers. In: Yu, P.S., Chen, A.L.P. (eds.) Int. Conf. Data Engineering. pp. 298–305. IEEE Computer Society (1995)
9. Lopez, V., Uren, V., Motta, E., Pasin, M.: Aqualog: An ontology-driven question answering system for organizational semantic intranets. Journal of Web Semantics 5(2), 72–105 (2007)
10. Montague, R.: Universal grammar. Theoria 36, 373–398 (1970)
11. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. In: *et al*, I.F.C. (ed.) Int. Semantic Web Conf. pp. 30–43. LNCS 4273, Springer (2006)
12. Schwitter, R., Kaljurand, K., Cregan, A., Dolbear, C., Hart, G.: A comparison of three controlled natural languages for OWL 1.1. In: Clark, K., Patel-Schneider, P.F. (eds.) Workshop on OWL: Experiences and Directions (OWLED). vol. 258. CEUR-WS (2008)
13. Smart, P.: Controlled natural languages and the semantic web. Tech. rep., School of Electronics and Computer Science University of Southampton (2008), http://eprints.ecs.soton.ac.uk/15735/